CS321: Computer Networks

Socket Programming

Dr. Manas Khatua
Assistant Professor
Dept. of CSE
IIT Jodhpur
E-mail: manaskhatua@iitj.ac.in

Socket Programming

- It shows how the network application programs are actually created
- typical network application consists of
  - a client program and a server program
  - Those programs resides in two different end systems.
- There are two types of network applications
  - Open i.e. operation rules are known to all
  - Proprietary i.e. operation rules has not been published
- In Server Site:
  - The server needs a local (server) and a remote (client) socket address for communication.
- In Client Site:
  - The client also needs a local (client) and a remote (server) socket address for communication.

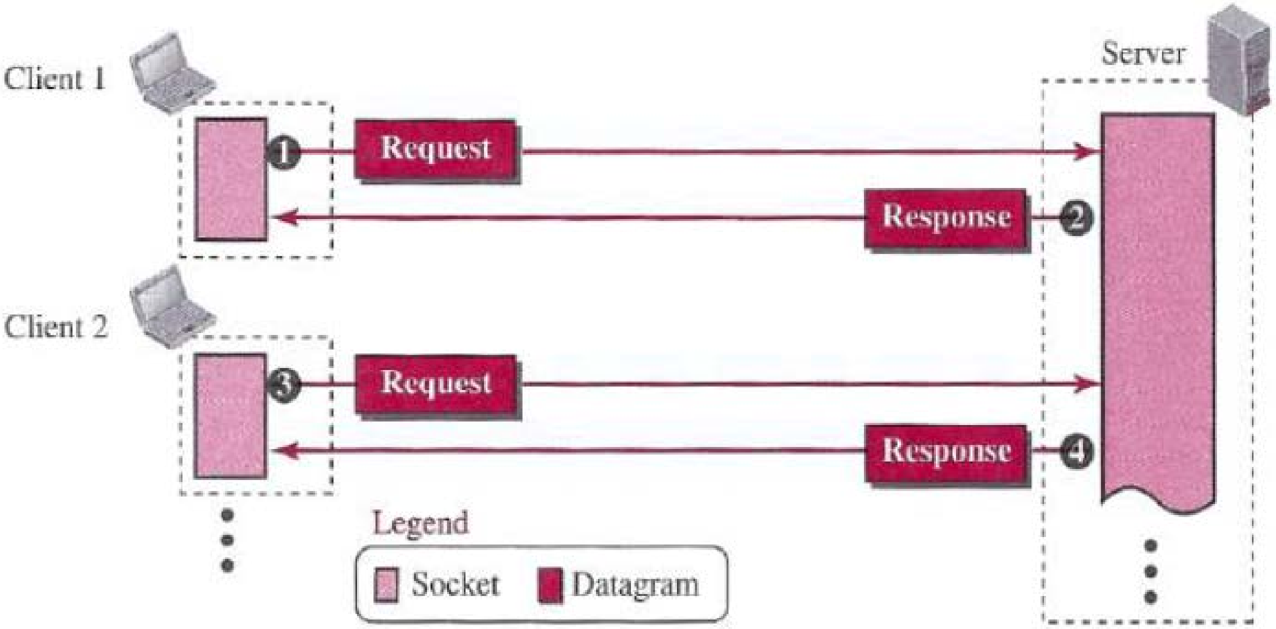
Iterative Communication Using UDP

- several client programs can access the same server program at the same time

- the server program can be designed to respond
  - Iteratively (i.e. one by one)
  - Concurrently

- In UDP communication, the client and server use only one socket each
  - The socket created at the server site lasts forever;
  - the socket created at the client site is closed (destroyed) when the client process terminates.

Cont…



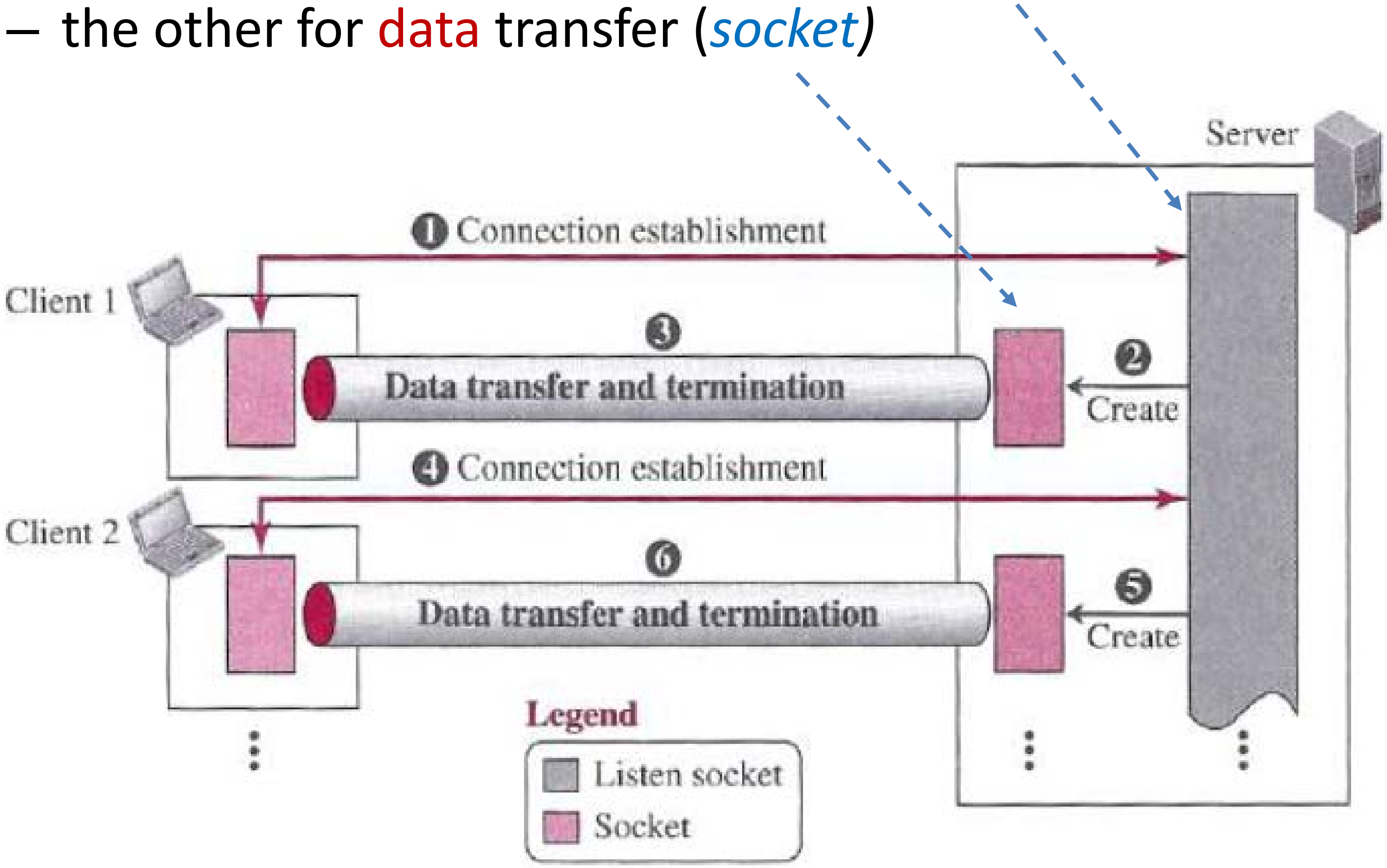
- Each client is served in each iteration of the loop in the server.

- Each client sends a single datagram and receives a single datagram.

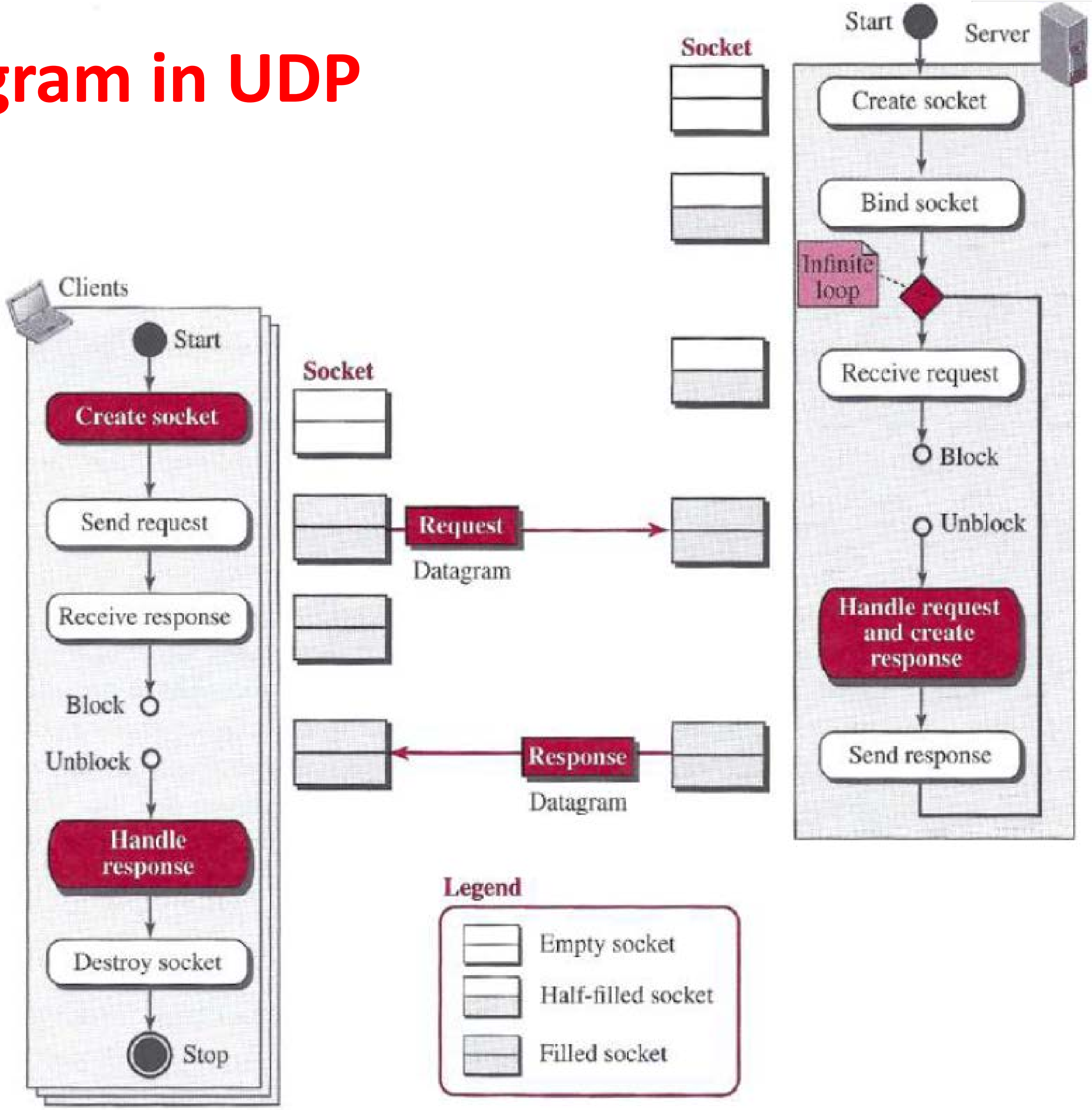
- If a client wants to send two datagrams, it is considered as two clients for the server.

Iterative Communication Using TCP

- The TCP server uses two different sockets
  - one for connection establishment (*listen socket)*
  - the other for data transfer (*socket)*

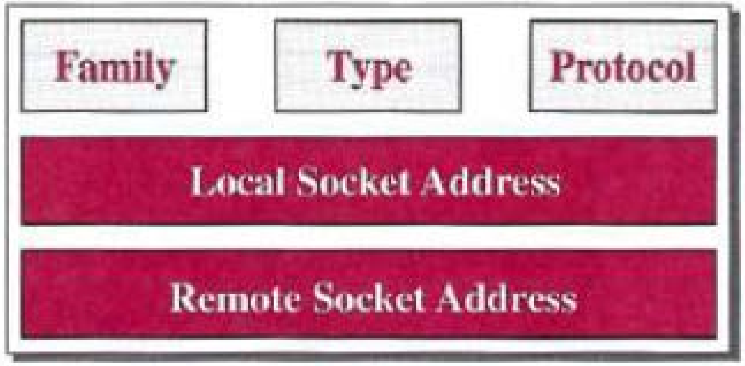

Server
1 Connection establishment
Client 1
3
Data transfer and termination
2
Create
4 Connection establishment
Client 2
6
Data transfer and termination
5
Create
Legend
Listen socket
Socket

Flow Diagram in UDP



Socket Programming (in C)

- the role of a socket in communication
  - has no buffer to store data to be sent or received
  - is capable of neither sending nor receiving data
  - acts as a reference or a label
  - buffers and necessary variables are created inside OS
- The C language defines a socket as a structure.



Socket

Cont…

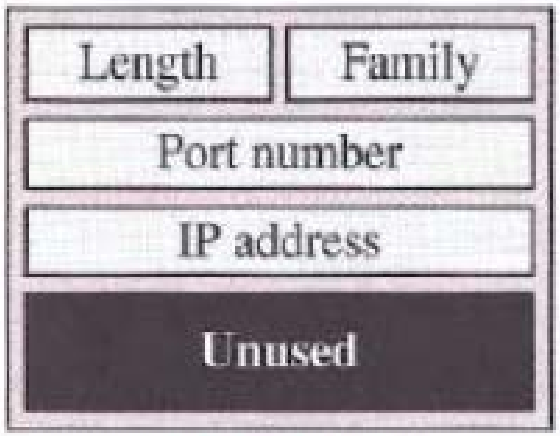
- *Family:* defines the protocol family (PF). The common values are PF_INET (for current Internet),

- *Type:* defines four types of sockets
  - SOCK_STREAM (for TCP),
  - SOCK_DGRAM (for UDP),
  - SOCK_SEQPACKET (for SCTP),
  - SOCK_RAW (for directly use the IP)

- *Protocol:* defines the specific protocol suite in the family said above (e.g., 0 => TCP/IP)

Cont…

- *Local socket address:* defines the *local socket address*
  - *Length: size of the socket address*
  - *Family:* AF_INET for TCP/IP protocol suite; *(AF: address family)*
  - *Port Number: it defines the process*
  - *IP Address: it* defines the host on which the process is running
  - *Unused: for future use*



Socket address

- *Remote socket address:* defines the remote socket address

Echo application using UDP

- Echo Application:
  - The client program sends a short string of characters to the server;
  - the server echoes back the same string to the client.

- "headerFiles.h" : need to use the definition of the socket and all procedures (functions) defined in the interface,

Socket

```
struct sockaddr_in {
    short               sin_family;       // e.g. AF_INET
    unsigned short      sin_port;         // e.g. htons(3490)
    struct in_addr      sin_addr;         // see struct in_addr, below. It is IP address.
    char                sin_zero[8];      // zero this if you want to
};
struct in_addr {
    unsigned long       s_addr;           // load with inet_aton()
};

struct   sockaddr_in   myaddr;
int s;
myaddr.sin_family = AF_INET;
myaddr.sin_port = htons(3490);            // host to network short
inet_aton("63.161.169.137", &myaddr.sin_addr.s_addr);
                //inet_aton::   convert from a struct in_addr to a string in dots-and-numbers

s = socket(PF_INET, SOCK_STREAM, 0);
bind(s, (struct  sockaddr*)myaddr, sizeof(myaddr));
```

Family
Type
Protocol

Echo Server Program

- // **UDP** echo server program

```
#include "headerFiles.h“        // All header files required for socket programming


int main (void)
{       // Declare and define variables
        int     s;                                          // Socket descriptor (reference)
        int     len;                                        // Length of string to be echoed
        char  buffer [256];                                 // Data buffer
        struct  sockaddr_in     servAddr;       // Server (local) socket address
        struct  sockaddr_in     clntAddr;       // Client (remote) socket address
        int      clntAddrLen;                               // Length of client socket address


        // Build local (server) socket address
        memset (&servAddr, 0, sizeof (servAddr));                  // Allocate memory
        servAddr.sin_family   = AF_INET;                            // Default Family field
        servAddr.sin_port      = htons (SERVER_PORT)               // Default port number
        servAddr.sin_addr.s_addr    = htonl (INADDR_ANY);           // Default IP address
```

Cont...

```
    // Create socket
    if ((s = socket (PF_INET, SOCK_DGRAM, 0) < 0);
    {       perror ("Error: socket failed! ");
            exit (1);
    }

    // Bind socket to local address and port
    If   (bind (s, (struct  sockaddr*) &servAddr, sizeof (servAddr)) < 0);
    {       perror ("Error: bind failed!");
            exit (1);
    }

    for ( ; ; )             // Run forever
    {       // Receive String
            len = recvfrom (s, buffer, sizeof (buffer), 0,
                                    (struct sockaddr*)&clntAddr,  &clntAddrLen);
            // Send String
            sendto (s, buffer, len, 0, (struct sockaddr*)&clntAddr, sizeof(clntAddr));
    } // End of for loop
} // End of echo server program
```

Echo Client Program

- //UDP echo client program

```
#include "headerFiles.h“

int main (int argc, char* argv[ ])    // Three arguments to be checked later
{     //Declare and define variables
      int      s;                     // Socket descriptor
      int      len;                   // Length of string to be echoed
      char* servName;                 // Server name
      int      servPort;              // Server port
      char*  string;                  // String to be echoed
      char    buffer [256+ 1];        // Data buffer
      struct   sockaddr_in   servAddr;   // Server socket address

      // Check and set program arguments
      if(argc !=3)
      {
              printf ("Error: three arguments are needed!");
              exit(1);
      }
      servName          =  argv[1];
      servPort          =  atoi (argv[2]);
      string            =  argv[3];
```

Cont…

```
// Build server socket address
memset (&servAddr, 0, sizeof (servAddr));
servAddr.sin_family = AF_INET;

//call DNS to find the server IP corresponding to server name
inet_pton (AF_INET, servName, &servAddr.sin_addr);
servAddr.sin_port = htons (servPort);

// Create socket
If ((s = socket (PF_INET, SOCK_DGRAM, 0) < 0);
{
	perror ("Error: Socket failed!");
	exit (1);
}
//Send echo string
len = sendto (s, string, strlen (string), 0, (struct  sockaddr)&servAddr, sizeof(servAddr));

//Receive echo string
recvfrom (s, buffer, len, 0, NULL, NULL);
// NULL: as we don’t need client socket address and length
```

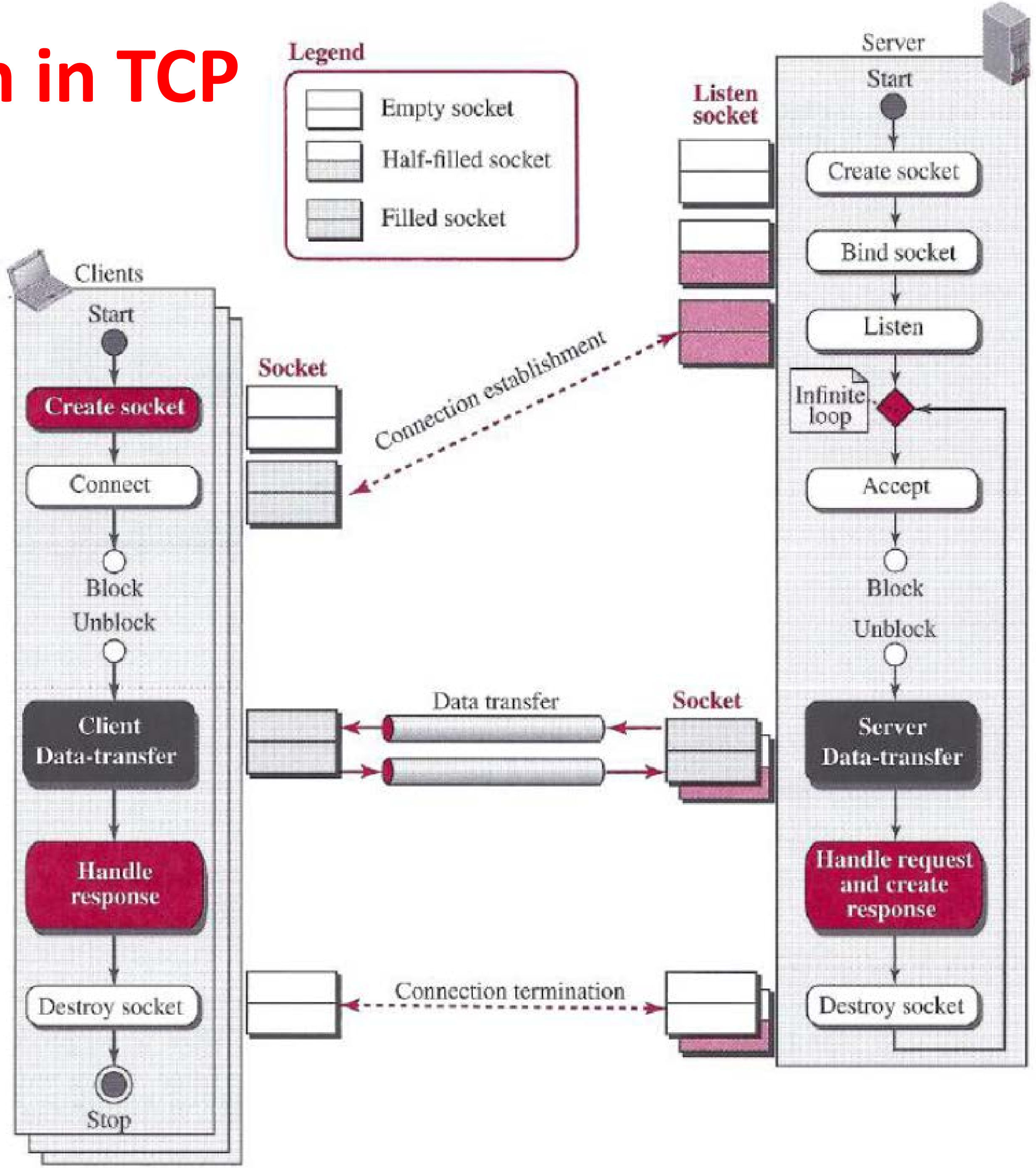
Cont…

```
    //Print and verify echoed string
    buffer [len] = '\0';
    printf ("Echo string received: “);
    fputs (buffer, stdout);

    //Close the socket
    close (s);

    //Stop the program
    exit (0);
} // End of echo client program
```

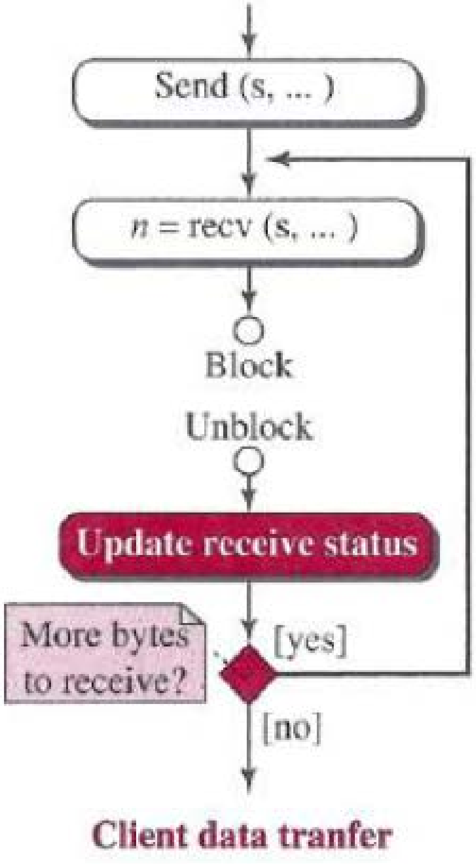
Flow Diagram in TCP

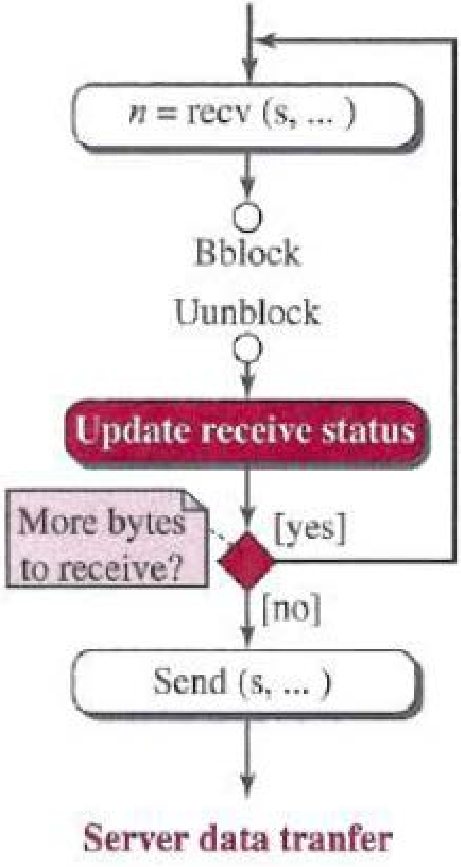


Iterative Programming Using TCP

- Before sending or receiving data, a connection needs to be established between the client and the server.

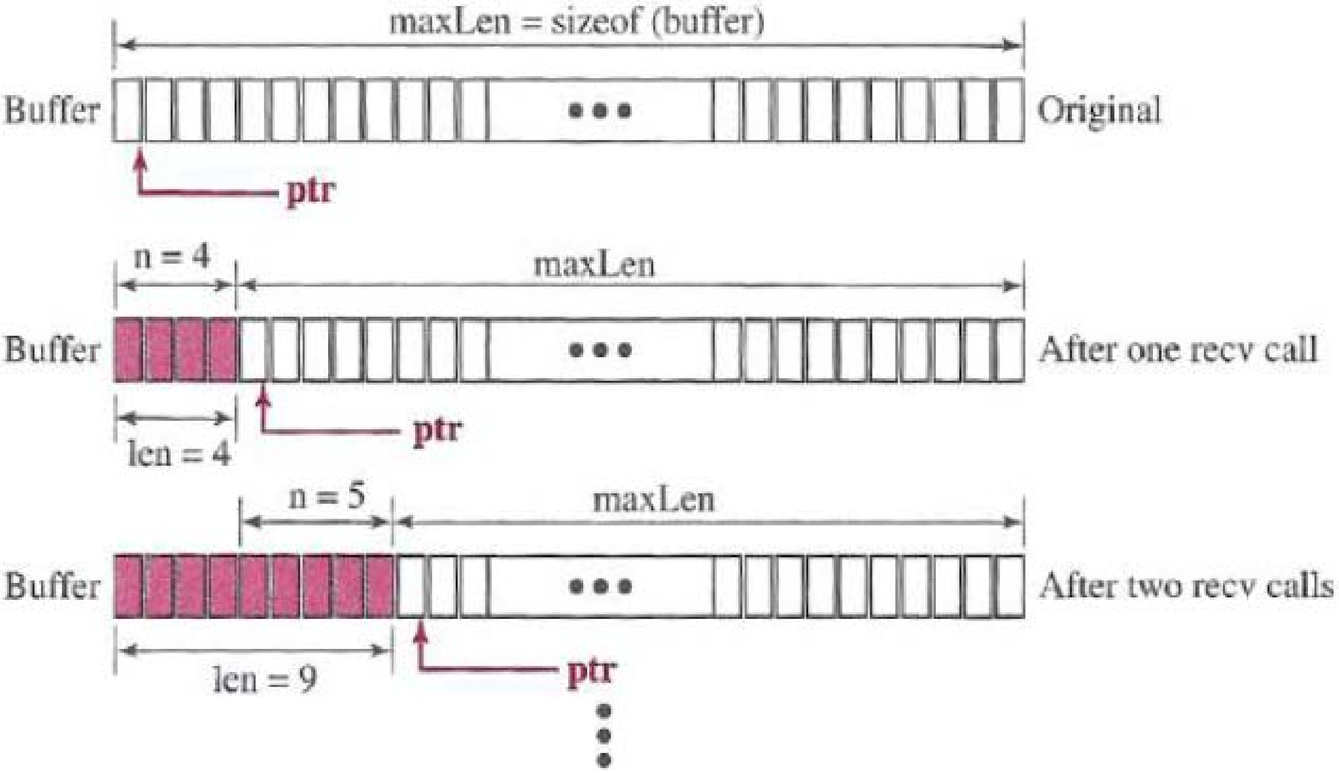
Flow diagram for the client and server data-transfer boxes





Cont…

- We need to control
  - how many bytes of data we have received and
  - where the next chunk of data is stored.



Echo server program using TCP

```
// Echo server program
#include "headerFiles.h“

int main (void)
{     // Declare and define
      int    ls;                            // Listen socket descriptor
      int    s;                             // Socket descriptor (reference)
      char  buffer [256];                   // Data buffer
      char*           ptr = buffer;         // Data buffer
      int    len = 0;                       // Number of bytes to send or receive
      int    maxLen = sizeof (buffer);      // Maximum number of bytes
      int    n = 0;                         // Number of bytes for each recv call
      int    waitSize = 16;                 // Size of waiting clients
      struct sockaddr _in   serverAddr;     // Server address
      struct sockaddr_in    clientAddr;     // Client address
      int    clntAddrLen;                   // Length of client address
```

Cont…

```
// Create local (server) socket address
memset (&servAddr, 0, sizeof (servAddr));
servAddr.sin_family = AF_INET;
servAddr.sin_addr.s_addr = htonl(INADDR_ANY);           // Default IP address
servAddr.sin_port = htons (SERV_PORT);                   // Default port

// Create listen socket
if (Is = socket (PF_INET, SOCK_STREAM, 0) < 0)
{        perror ("Error: Listen socket failed!");  exit (1);    }

// Bind listen socket to the local socket address
if (bind (Is, &servAddr, sizeof (servAddr)) < 0)
{        perror ("Error: binding failed!");  exit (1);          }
// Listen to connection requests
if (listen (Is, waitSize) < 0)
{        perror ("Error: listening failed!");  exit (1);        }
```

Cont…

```
// Handle the connection
for ( ; ; )   // Run forever
{
    // Accept connections from client
    if (s = accept (ls, &clntAddr, &clntAddrLen) < 0)
    {       perror ("Error: accepting failed!);         exit (1);      }

     // Data transfer section
     while ((n = recv (s, ptr, maxLen, 0)) > 0) {
          ptr += n ;          // Move pointer along the buffer
          maxLen -= n; // Adjust maximum number of bytes to receive
          len += n;          // Update number of bytes received
     }
     send (s, buffer, len, 0);  // Send back (echo) all bytes received
     close (s); // Close the socket
} // End of for loop
} // End of echo server program main
```

Echo client program using TCP

```
// TCP echo client program
#include "headerFiles.h“

int main (int argc, char* argv[ ])
{     // Declare and define
      int    s;                    // Socket descriptor
      int    n;                    // Number of bytes in each recv call
      char*  servName;             // Server name
      int    servPort;             // Server port number
      char*  string;               // String to be echoed
      int    len;                  // Length of string to be echoed
      char  buffer [256 + 1];      // Buffer
      char*   ptr = buffer;        // Pointer to move along the buffer
      struct  sockaddr_in    serverAddr;    // Server socket address

      // Check and set arguments
      if (argc != 3)
      {         printf ("Error: three arguments are needed!");
                 exit (1);
      }
```

Cont…

```
servName = arg [1];
servPort    = atoi (arg [2]);
string        = arg [3];

// Create remote (server) socket address
memset (&servAddr, 0, sizeof(servAddr);
serveAddr.sin_family = AF _INET;
inet_pton (AF_INET, servName,&serveAddr.sin_addr);         // Server IP
serveAddr.sin_port = htons (servPort);                      // Server port

//Create socket
if ((s = socket (PF_INET, SOCK_STREAM, 0) < 0);
{   perror ("Error: socket creation failed!");
     exit (1);  }

// Connect to the server
if (connect (sd, (struct sockaddr*)&servAddr, sizeof(servAddr)) < 0)
{     perror ("Error: connection failed!");
       exit (1);   }
```

Cont…

```
    // Data transfer section
    send (s, string, strlen(string), 0);

    while ((n = recv (s, ptr, maxLen, 0)) > 0)
    {
        ptr + = n;                      // Move pointer along the buffer
        maxLen - = n;                   // Adjust the maximum number of
        len += n;                       // Update the length of string received
    }

    // Print and verify the echoed string
    buffer [len] = '\0';
    printf ("Echoed string received: ");
    fputs (buffer, stdout);

    // Close socket
    close (s);

    // Stop program
    exit (0);
} // End of echo client program main
```

Thanks!